Web-Server: Hochleistungs-Apache: Performance-Tuning

Apache am Ende?

Apache gilt als der beste Webserver für Linux/Unix. Ist er wahrscheinlich auch. Gerade in der Version 2, bei der deutlich mehr Konfigurationen möglich sind. Aber sobald der Indianer eine Menge zu tun bekommt kann er schon mal das ganze System in die Knie zwingen. Daher hier ein paar Tipps für ausgelastete Apache.

Bitte bedenkt aber, daß die Optimierung immer ein dynamischer Prozess ist! (Siehe Punkt e.)

Apache aufbohren, tieferlegen, tunen

Es sind zwei wesentliche Dinge, die den Apache beschleunigen und damit schnelleres Arbeiten erlauben:

a) Speicherverbrauch minimieren.

b) Speicher besser aufteilen und Requests beschränken.

c) Finetuning

d) Testing

e) Beobachtung

Die erste Station lässt sich meistens leicht entscheiden, ob man gewisse Module braucht oder nicht.

Schwieriger wird es bei Punkt b) und c). Hier gibt es keine allgemeine Aussagen á la "Bei XX-Usern in YY-Forum auf ZZ-Hardware braucht man folgende Einstellungen." Jeder der so etwas behauptet (oder hier sucht) liegt Grundsätzlich falsch.

Der Tuning-Prozess ist immer eine eigenständige Entwicklung und muß an die Gegebenheiten angepasst werden. Hier braucht man etwas Experimentierfreudigkeit und Geduld.

a) Entschlackung:

Der Apache kann mit Modulen vielseitig erweitert werden. Nachteil: Die Module nehmen Speicher und Performance in Anspruch. Denn jeder Request durchläuft fast alle Module, welche jeweils prüfen, ob es damit etwas zu tun hat. D.h. daß auch pro Request von jedem Modul Speicher angefordert wird.

Die vorinstallierte Apache-Conf der jeweiligen Linux-Distribution ist für die "breite Masse" ausgelegt und i.d.R. nicht für den jeweiligen in Betrieb befindlichen Server.

Hier ein kleiner Auszug aus möglichen Modulen die Sinn oder weniger Sinn haben:

- Braucht man Server-Auth? Wenn ja welche?

  Meistens wird nur `mod_auth` benötigt.

- Dynamische Seiten:

  + mod_log_config erlaubt die access_log-Dateien in div. Formaten.

  + mod_cgi braucht man, wenn man CGI zulassen will.

  + mod_suexec erlaubt CGIs als ein bestimmter User zu laufen.

  - mod_perl braucht man nicht um Perl als CGI auszuführen!

  + mod_php4 (5) ist schneller als PHP als CGI (aber nicht so sicher).

  - mod_include erlaubt die Verwendung von SSI.

- URL-Manipulationen:

  + mod_alias braucht man zum Einblenden von virtuellen Verzeichnissen.

  - mod_rewrite erlaubt URL-Manipulationen, kostet aber Performance.

  - mod_autoindex & mod_dir erlauben eine Navigation ohne index.html

  - mod_userdir erlaub User-URLs mit z.B.: www.huschi.net/~huschi/

  - mod_negotiation analysiert Accept-Anfragen.

  + mod_vhost_alias erlaubt das Anlegen von virtuellen Hosts.

- Environment- & Header-Manipulation:

  + mod_mime setzt automatisch den richtigen Header anhand der Dateiendung.

  - mod_env & mod_setenvif erlauben die modifizierung von ENV-Variablen.

  - mod_expires erlaubt die HTTP-Header Manipulation des Expires-Date.

  - mod_ssl erlaubt die Verwendung von SSL (https).

b) MPM Tuning:

Beispiel für ein MPM-Prefork welcher knapp 500 (Massenhosting-)Webs auf einen 2GHz-Server mit 2 GByte Speicher verwaltet inkl. mod_php, CGI, rewrite, etc.:

```
StartServers          5
MinSpareServers       5
MaxSpareServers      10
ServerLimit         150
MaxClients          150
MaxRequestsPerChild   0
```

Der Parameter `MaxClients` entspricht den maximal simultan abzuarbeitenden Request. Er kann nicht höher als `ServerLimit` sein. Wenn diese Anzahl an simultanen Prozessen ausgeschöpft ist, werden weitere Requests nicht verworfen sondern ein eine Warteschlange (queue) eingereiht welche dann entsprechende längere Antwortzeiten haben.

Im Laufe der Zeit hat sich eine Faustregel für den Prefork-MPM manifestiert:

Ein Apache-Prozess benötigt rund 12 MB Speicher. 12 MB mal MaxClients ist dann der Speicher den man (ohne Swap) frei haben sollte.

Normale Root-Server und vorallem virtuelle Server sollten den `MaxClients`-Wert möglichst gering halten. Hier ist `mod_status` zur Analyse des realen Bedarfs sehr Hilfreich.

Zusätzlich hat hier ein `KeepAlive Off` den Server vor dem Abrauchen bewart.

Wer mag, kann natürlich versuchen den KeepAlive mit einem entsprechend kleinen TimeOut (hier 2 Sekunden) zu setzen:

```
KeepAlive On
KeepAliveTimeout 2
MaxKeepAliveRequests 10
```

c) Finetunig:

- Vermeiden von DNS-Lookups: `HostNameLookups off`

  Denn erstmal jede Client-IP auf seinen Hostnamen aufzulösen ist unnötig und kostet nur Zeit.

- Vermeiden von .htaccess-Dateien:

  Diese Dateien werden pro Request geprüft! Und zwar den ganzen Verzeichnispfad von `DocumentRoot` bis zum Ziel-Verzeichnis.

  Die Direktiven von dort sollte man also am besten in die Apache-Config übernehmen und mit `AllowOverride None` sucht Apache erst gar nicht nach `.htaccess`-Dateien.

d) Performance-Test:

Hierfür bietet sich das von Apache mitgelieferte `ab2` an. Damit kann man einige Requests in Serie abfeuern und erhält direkt eine statistische Auswertung.

Auch hier gilt: Es gibt keine Normwerte die ein Server erfüllen muß! Das Tool sehe ich eher als Erfolgstest an, um die neuen Parameter zu testen. Es simuliert auch keine echten Benutzer/Surfer auf dem Server.

e) Beobachtung

Weitere Optimierungsschritte finden sich schnell, wenn man den Server vernünftig beobachtet. Dazu nutzt man `mod_status` im `ExtendedStatus On` Modus.

Besondere Beachtung sollten Scripte (PHP/Perl/etc.) haben. Sollte hier vereinzelt die CPU-Zeit oder Laufzeit extrem hoch gehen, wird eine gewisse Aktivität erforderlich.

weiter Links:

- huschi.net: Apache-Optimierung auf HTTP-Ebene
- Apache HTTP-Server: Modul-Index
- Linux-Magazin 2004/01: Häuptling Schnelles Wiesel
- Linux-NetMag: Configuring Apache for Maximum Performance (2006)
  (Link verlegt auf berlios.de)

Eindeutige ID: #1053
huschi
2009-06-29 14:13